L’informatica e i linguaggi di programmazione

A cura del docente

Gaetano Carpinato

Cos’è l’Informatica

Informatica = Informazione (dati + istruzioni) + Automazione

L’informatica è la scienza che studia il computer sotto tutti i suoi aspetti. In inglese si usa l’espressione “computer science”, mentre in italiano si preferisce la parola “informatica”, che nasce dalla contrazione delle parole *informazione* e *automatica*, per indicare l’elaborazione automatica dell’informazione, che è appunto l’attività effettuata dal computer.

L’informatica è una scienza che non si occupa di oggetti naturali (a differenza di fisica, chimica e biologia) o di oggetti ideali (a differenza della matematica), bensì di oggetti sociali, costruiti dagli uomini secondo un progetto e per degli scopi ben precisi. In tal senso l’informatica risulta affine a discipline come il diritto, l’economia, l’italiano o il disegno, pur essendo una disciplina scientifica come matematica, fisica, chimica. Pertanto l’informatica si può considerare come un sistema di regole che devono essere stabilite e seguite perché l’uso del computer sia efficace.

Per capire come funzionano i computer, occorre quindi capire che cos’è un sistema di regole e che cosa significa seguirle. Le regole sono leggi che vengono stabilite all’interno di una comunità e che le persone accettano di seguire per raggiungere determinati obiettivi. L’informatica è un sistema di regole per la costruzione, il funzionamento e l’uso dei computer.

Soltanto se siamo in grado di seguire le regole sociali possiamo imparare le regole informatiche. Dunque i due momenti sono parte di un’*unica valutazione*.

Cos'è il Computer

Informazione = Dati + Istruzioni

È un insieme organizzato di componenti, in grado di eseguire una successione di *istruzioni*, su un insieme di *dati* di partenza, finalizzate a elaborare *l'informazione*. È in grado di eseguire operazioni relativamente semplici ad alta velocità.

La differenza fondamentale fra il computer e i suoi antenati (primo fra tutti la calcolatrice) è che il computer è in grado di ricevere e memorizzare non soltanto semplici **dati** (numeri, testi, immagini, suoni) ma anche **istruzioni** (azioni da compiere sui dati). Ne consegue che il computer è una macchina programmabile, in grado di eseguire una quantità illimitata di elaborazioni diverse (anziché una quantità limitata come i tasti che corrispondono alle operazioni della calcolatrice). Dunque nel computer le istruzioni sono una parte fondamentale del processo di esecuzione per ricavare le informazioni (il risultato).

Elaborare le *informazioni*, che è l'attività propria del computer, significa ricevere dei *dati* e produrre dei risultati(*informazioni*). I dati ricevuti si chiamano **input**, i risultati prodotti si chiamano **output**. Un programma/algoritmo a questo punto si può definire come una sequenza di istruzioni che trasformano l'input(*dati*) nell'output(*informazioni*).

Ricapitolando:

Input, cioè Dati (di partenza, in ingresso)

+

Istruzioni

=

Output, cioè Informazione o Risultato

L'utilizzatore che è in grado di introdurre nella macchina soltanto dati (affidandosi a elaborazioni predefinite) si chiama **utente**; l'utilizzatore che è in grado di introdurre anche istruzioni (e quindi di dire al computer non solo che cosa fare, ma anche come farlo) si chiama **programmatore**.

Utente e programmatore sono due ruoli diversi e bisogna tenerne conto anche quando ad interpretarli è la stessa persona.

Il computer visto come l'essere umano

Si può paragonare il computer all'essere umano e quindi l'informatica (la scienza che studia il computer) alle scienze che studiano l'essere umano.

Le componenti del computer si suddividono in:

- **Hardware:** è il corrispettivo del corpo umano ed è la componente materiale del computer. Comprende tutto ciò che nel computer occupa una porzione di spazio fisico e quindi, in linea di massima, si può vedere e toccare
- **Software:** è invece il corrispettivo della mente intesa come pensieri, cioè la componente astratta del computer. Comprende tutte le informazioni che il computer elabora e tutte le istruzioni con cui le elabora
- **Periferiche di Input:** è il corrispettivo della percezione (i dati che riceviamo dal mondo)
- **Periferiche di Output:** è il corrispettivo dell'azione (i risultati che produciamo nel mondo)

Il paragone fra computer e essere umano è utile per chiarire alcuni concetti, ma va trattato con l'opportuna cautela. La differenza fondamentale è che nel computer il software è separabile dall'hardware (e riproducibile), mentre nell'essere umano – almeno allo stato attuale delle conoscenze scientifiche – la mente non è separabile dal corpo (né riproducibile).

I componenti hardware di un computer

Memoria centrale: è lo spazio fisico in cui vengono immagazzinate le *istruzioni* che compongono i programmi, i *dati* immessi dall'utente e le *informazioni* ricavate dalle operazioni. La memoria centrale viene suddivisa in due componenti:

- RAM (Random Access Memory): memoria "volatile" o memoria di lavoro. In essa vengono conservati(*mantenuti*) i dati in corso di elaborazione e le istruzioni del programma in esecuzione (perciò tutte le informazioni durante un'intera sessione di lavoro). Allo spegnimento del computer tutto ciò che è contenuto nella RAM viene perso (per questo viene definita volatile).
- ROM (Read Only Memory): la parte non volatile della memoria centrale, serve nella fase di accensione del computer (*bootstrap*). Contiene parti essenziali del software di sistema quali il *BIOS*.

CPU (Central Processing Unit): detto processore in italiano, è il "cervello" del computer ed è composta da un'Unità di Controllo (CU) che governa il funzionamento della macchina e gestisce le relazioni con memoria(RAM) e CPU e da un'Unità di Calcolo (ALU – Arithmetic Logic Unit) che esegue operazioni aritmetiche e logiche.

Periferiche di comunicazione: si definiscono tali i dispositivi che permettono di svolgere le operazioni di *input* (tastiera, mouse) e di *output* (schermo, stampante). Il modem è una periferica di comunicazione che serve a comunicare, in ambo le direzioni(*input/output*), con un altro computer anziché con l'utente.

Periferiche di memorizzazione(*storage*) o memorie secondarie: si definiscono tali i dispositivi (hard-disk, chiavette, CD, DVD) che permettono di sopperire alle dimensioni limitate e alla natura "volatile" della memoria centrale.

Cos'è un algoritmo

Sequenza **finita** e **ordinata** di passi/operazioni che portano alla realizzazione di uno specifico compito. Può essere vista come una "ricetta" che spiega all'esecutore come produrre i risultati a partire dai dati. L'algoritmo di un'operazione complessa può essere scomposto in una sequenza di istruzioni più semplici. L'algoritmo ha **sempre** carattere generale: non risolve un singolo problema (i.e. quanto fa 1/3 + 2/5), ma una famiglia di problemi (i.e. come si sommano due frazioni).

Esempi: calcoli matematici, massimo comune divisore, istruzioni di un elettrodomestico, prelevamento Bancomat, ricette di cucina.

N.B. Un computer è un esecutore di algoritmi

Proprietà di un algoritmo: **correttezza** (deve giungere alla soluzione del dato problema) ed **efficienza** (dare la soluzione nel modo più veloce, utilizzando la minima quantità di risorse fisiche).

Metodi di rappresentazione di un algoritmo (formalismi):
- Linguaggio naturale
- Diagramma di flusso / flow chart
- Pseudo-codice
- Linguaggio di programmazione

Nei linguaggi di programmazione, un passo/operazione viene chiamata **istruzione**.

Alcuni esempi di *istruzioni* sono:

– Operazioni di Input/Output (es. leggi, scrivi)
– Operazioni Aritmetico-logiche (es. somma = A + B)
– Strutture di controllo, che vengono definiti **costrutti.** I costrutti possono essere di *selezione* (SE-ALTRIMENTI) o *ciclici* (RIPETI)

Cos'è un programma

Un programma è la traduzione di un insieme di uno o più algoritmi (*scritti in un determinato linguaggio di programmazione*) in un linguaggio comprensibile dal computer (*linguaggio macchina*). Il processore del

computer esegue i programmi passo-passo in modo preciso e veloce. L'insieme dei **dati** e delle **istruzioni** finalizzate alla produzione di un risultato o di uno scopo ben preciso si chiama programma.

Programmare significa risolvere problemi col computer, cioè far risolvere problemi al computer attraverso un insieme di dati e una sequenza di istruzioni. La programmazione si compone di tre fasi: analisi del problema (*consiste nell'individuare i dati e i risultati del problema che bisogna risolvere*), scrittura degli algoritmi, creazione del programma.

Gli algoritmi sono parametrici

- Producono un risultato(*informazione*) che dipende da un insieme di dati di partenza

- Descrivono la soluzione non di un singolo problema, ma di una intera classe di problemi strutturalmente equivalenti

Esempi:

- L'algoritmo per la moltiplicazione di due numeri specifica come effettuare il prodotto di tutte le possibili coppie di numeri

- L'algoritmo per la ricerca di un libro nello schedario della biblioteca vale per tutti i possibili libri

Le istruzioni dell'algoritmo fanno riferimento a **variabili**, il cui valore non è fissato a priori ma cambia a seconda della situazione elaborativa in cui il processore del computer si trova.

Cos'è una variabile

Una variabile è un **dato**, caratterizzato dal *tipo*, un'*etichetta* e un *valore*. Possiamo pensare a una variabile come ad un contenitore dentro la quale possiamo (generalmente) leggere e scrivere, mantenuta all'interno della memoria RAM. Questi contenitori possono essere riempiti con un *valore* che poi può essere riletto oppure sostituito. Nei linguaggi di programmazione le variabili possono essere di diverso tipo: ci sono variabili che possono contenere numeri interi, altre che possono

contenere numeri con la virgola, altre possono contenere stringhe o oggetti più complicati.

Formalmente ogni variabile è una porzione della memoria del computer(RAM) in cui è possibile memorizzare un ***valore*** *da utilizzare all'interno di un programma. Tutte le variabili devono avere un* ***nome*** *e un* ***tipo****.*

Il **tipo** di una variabile definisce quale tipo di dato (ad esempio numerico o stringa) possiamo scrivere al suo interno, come interpretare il dato memorizzato e lo spazio necessario alla sua memorizzazione all'interno della RAM. I tipi vengono suddivisi in

Tipi primitivi:

– Numeri naturali o interi o reali (1, -2, 0.34)
– Caratteri alfanumerici (A, B, a, c, 1, 2, -, |, ...)
- N.B. un insieme ordinato (l'ordine degli elementi ha importanza) di caratteri alfanumerici viene definita stringa (*il fatto che gli elementi dell'insieme abbiano un ordine non significa che gli elementi siano ordinati tra loro*)

– Dati logici o booleani (*true*, *false*)

Struttura dati astratta:

– Array o vettore di n elementi: {0, 1, 2}
- N.B. un array/vettore è un insieme ordinato (l'ordine degli elementi ha importanza) di elementi di un dato tipo (*il fatto che gli elementi dell'array abbiano un ordine non significa che gli elementi siano ordinati tra loro*). Essendo ordinato, è possibile accedere ad un elemento dell'array/vettore attraverso il suo **indice**, che corrisponde alla posizione dell'elemento all'interno dell'array/vettore

– Matrice di n per m elementi:
{ {0-0, 0-1, 0-2},
{1-0, 1-1, 1-2},
{2-0, 2-1, 2-2},
{3-0, 3-1, 3-2} }
- N.B. una matrice non è altro che un vettore di vettori

– Stringhe: Una **stringa** è un particolare array/vettore. Infatti è un insieme ordinato (stesso discorso di prima) di singoli caratteri alfanumerici
– Oggetti (li vedremo più avanti)

L'etichetta è il nome, *unico nel suo scope**, con cui riferirsi alla variabile all'interno del linguaggio di programmazione. In generale i linguaggi di programmazione distinguono minuscolo e maiuscolo (**case-sensitive**). I nomi delle variabili si scelgono solitamente in modo che sia facile intuire il loro contenuto. Deve rispettare le seguenti regole sintattiche: deve iniziare con una lettera minuscola o il simbolo underscore (_) e può contenere lettere minuscole/maiuscole, cifre da 0 a 9 e il simbolo underscore (_).

Il **valore** è l'informazione che è effettivamente scritta dentro la casella in un certo istante, durante l'esecuzione del nostro programma. Il tipo di valore che può assumere una variabile, in generale, dipende dal tipo della variabile. Quando si colloca un valore in una variabile, il valore precedentemente contenuto, se c'era, viene irrimediabilmente perduto.

*La **visibilità** (in inglese *scope*), in informatica, è l'esistenza e la possibilità di richiamare una variabile (o una funzione), in un determinato punto del programma. Una variabile visibile all'interno di un una sequenza di istruzioni raggruppate all'interno di un *costrutto* è in generale visibile anche all'interno di eventuali *costrutti* annidati.

Le variabili devono essere **dichiarate** prima di poter essere utilizzate, possono essere **inizializzate** all'atto della dichiarazione. All'atto della dichiarazione il tipo della variabile, se non è stato assegnato, è indefinito (**undefined**). L'inizializzazione assegna un valore ad una variabile e di conseguenza il tipo se non è stato definito precedentemente.

Esempio della dichiarazione di una variabile:

<tipo> nome;

Dove <tipo> è il tipo della variabile che stiamo dichiarando, nome è l'etichetta con la quale ci riferiremo alla variabile per poter scrivere/leggere il valore.

N.B. L'unico tipo di dato presente nel linguaggio di programmazione Javascript è var (sta per *variant*, si tratta di una tipologia di dati mista: una variabile dichiarata come variant può assumere vari tipi di dati: numeri interi, decimali, stringhe, booleani, oggetti e persino errori).

Javascript e PHP permettono di specificare solo variant come tipo della variabile, sarà quindi il valore che andremo ad inserire a definire il tipo effettivo. Pertanto Javascript e PHP vengono detti linguaggi debolmente tipizzati (non fortemente tipizzati).

Le regole fondamentali di visibilità sono in genere modificate dalla regola speciale dello shadowing, secondo cui una variabile "nasconde" una eventuale variabile omonima definita nello scope superiore. In altre parole, se in un blocco è definita una variabile e all'interno di un blocco annidato al precedente viene definita una variabile omonima, il blocco annidato in questione perde la visibilità della variabile più esterna, nascosta da quella appena dichiarata.

Esistono due ambiti di visibilità (scope) di un identificatore comuni a tutti i linguaggi di programmazione object-oriented:

A livello di blocco: Caratterizza le variabili *locali*; sono gli identificatori dichiarati in un blocco. Lo scope inizia dalla dichiarazione dell'identificatore e termina con la fine del blocco stesso. Le variabili locali dichiarate all'interno di una funzione, così come i parametri di una funzione, hanno visibilità a livello di blocco. Nel caso di blocchi nidificati, se un identificatore del blocco esterno ha lo stesso nome di quello del blocco interno, l'identificatore del blocco esterno viene occultato fino alla fine del blocco più interno secondo la regola dello shadowing. Il blocco interno vede solamente il proprio identificatore locale.

A livello di file: Caratterizza le variabili *globali*; un identificatore dichiarato all'esterno di una qualsiasi funzione ha visibilità a livello di file. È noto a tutte le funzioni che si trovano dopo la sua dichiarazione

I Sottoprogrammi o funzioni

Quando si scrive un programma succede spesso che si debba eseguire numerose volte una stessa sequenza di istruzioni, sugli stessi dati o su dati diversi.
Supponiamo di dover realizzare un programma per il gioco degli scacchi: ogni volta che si esegue una mossa i pezzi devono essere ridisegnati o nella stessa posizione o nella nuova posizione conseguente alla mossa. Per evitare di riscrivere più volte queste sequenze di istruzioni e per limitare le dimensioni dei programmi, i linguaggi di programmazione dispongono di una particolare struttura chiamata *funzione* o *sottoprogramma*.

Una **funzione** è un modulo di programma la cui esecuzione può essere invocata più volte nel corso del programma principale. La funzione può calcolare uno o più valori, che saranno comunicati al programma chiamante al termine della sua esecuzione, oppure può eseguire azioni generiche. Il nome funzione deriva dall'evidente analogia con le funzioni intese in senso matematico. La funzione può fornire uno e solo un valore al suo completamento attraverso l'istruzione:

return <variabile>;

L'istruzione *return* provoca la restituzione del controllo al blocco chiamante nel quale valore viene associato al nome della funzione che pertanto appare come una normale variabile.

L'elaborazione dei dati

L'elaborazione è l'insieme delle operazioni che permettono di passare dai dati alle informazioni. L'istruzione fondamentale per l'elaborazione è **l'assegnazione** che consiste nel calcolo di un'espressione e nella scrittura del risultato dentro una variabile.
La sua forma è questa:

<variabile> **=** <espressione>;

Per **espressione** si intende una combinazione di valori, variabili, operatori (+, -, *, /, %) ed eventuali parentesi (solo le tonde!) conforme alle regole di calcolo.

Uso delle variabili

- All'interno di espressioni (l'esecutore usa il valore contenuto nelle variabili per calcolare il risultato dell'espressione, per esempio op1 + op2 × op3 oppure op1 / op2 – op3, …)
- In istruzioni di assegnamento (introdurre nel contenitore identificato dal nome della variabile il valore specificato a destra dell'assegnamento, per esempio r = 35 (assegna 35 alla variabile il cui nome è r), pi = 3,14, …)
- In istruzioni di assegnamento combinate con espressioni (assegna a una variabile il risultato ottenuto dalla valutazione di un'espressione, per esempio in "circ = 2 × r × pi" il risultato dell'espressione 2 × r × pi viene calcolato utilizzando i valori contenuti nelle variabili r e pi e il risultato viene poi assegnato alla variabile circ; la stessa variabile può comparire in entrambi i lati dell'istruzione di assegnamento, per esempio in "k = k + 1" il valore contenuto in k viene utilizzato per trovare il valore dell'espressione k + 1 che viene memorizzato come nuovo valore di k.)

Assegnazione di valori a variabili

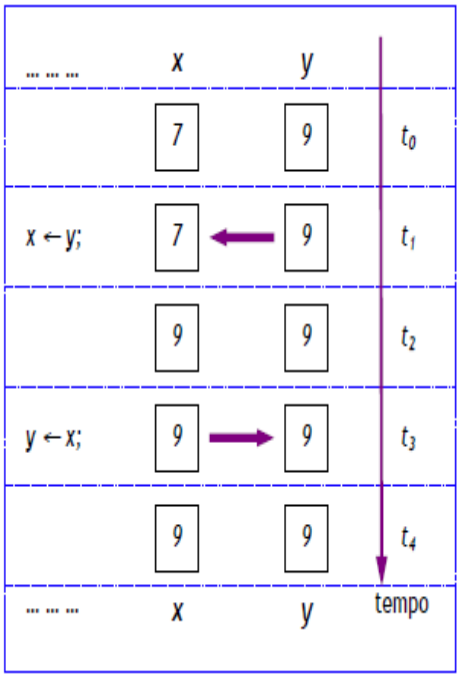
Il valore assegnato a una variabile si sostituisce a quello che era presente in precedenza, il vecchio valore non potrà più essere recuperato.

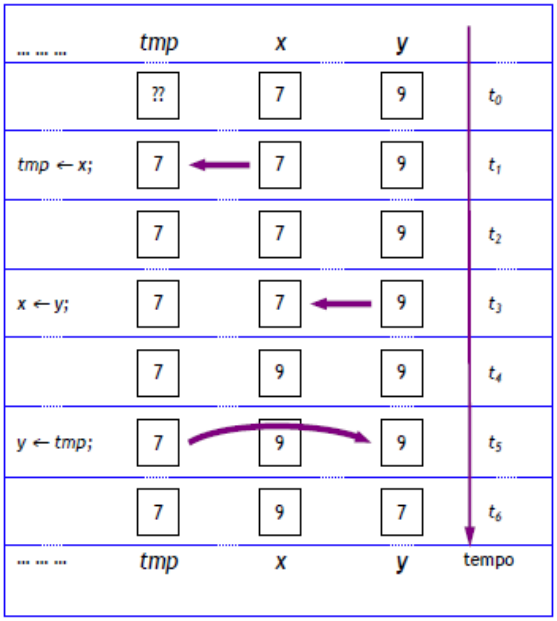
Esempio: si ipotizzi di voler scambiare i valori contenuti in due variabili x e y.

Soluzione proposta: doppio assegnamento del tipo

```
x = y;
y = x;
```

Per indicare che il valore di y deve essere copiato in x e che, nello stesso tempo, il valore di x sia trasferito in y. Le istruzioni però vengono eseguite in sequenza! Quindi l'assegnamento x = y viene completato prima di iniziare y = x.





Soluzione corretta: uso di una variabile aggiuntiva (tmp), come strumento di memorizzazione temporanea ("buffer") del valore originariamente contenuto in x

```
tmp = x;
x = y;
y = tmp;
```

In questo modo lo scambio avviene senza perdere i valori originari.

Le tre modalità di esecuzione di un programma

La modalità basilare di esecuzione di un programma è la **sequenza**: le istruzioni vengono eseguite una dopo l'altra, riga per riga, dall'alto in basso.

Le strutture di controllo modificano l'esecuzione sequenziale permettendo al programma di scegliere fra sequenze di istruzioni differenti (**selezione**) o di ripetere una sequenza di istruzioni (**iterazione**).

Il teorema di Böhm-Jacopini (1966) afferma che qualunque algoritmo può essere codificato usando soltanto *sequenza*, *selezione* e *iterazione*.

L'idea è che per spiegare a qualcuno che cosa deve fare, dobbiamo farlo procedere un passo alla volta, ma anche, in certi punti, farlo *scegliere* o *ripetere*.

Il costrutto selezione

La **selezione** è la struttura di controllo che permette all'algoritmo di decidere fra due *blocchi* di istruzioni, in base al valore di una condizione.

Un **blocco** è una sequenza di istruzioni raggruppate.

La **condizione** è una proposizione logica, cioè un'espressione il cui valore può essere *vero* oppure *falso*. Essa è generalmente basata sugli operatori di confronto (uguale, diverso, maggiore, minore…), Es. a > 3 è vera per a che vale 5 mentre è falsa per a che vale 1.

La condizione si scrive usando gli operatori aritmetici:

> , >= , < , <= , == , ===, !=

E se occorre anche gli operatori logici:

&& (AND), **||** (OR), **!** (NOT).

E le parentesi tonde **(** e **).**

Attenzione a non confondere **mai** l'operatore = (*assegnazione*) con l'operatore == (*confronto tra valori*) e === (*confronto tra valori e tipi*).

La forma algoritmica della selezione (pseudo-codice):

se (condizione) **allora** { blocco-1 }

altrimenti { blocco-2 }

Se la condizione è vera viene eseguito il blocco di istruzioni della condizione (blocco-1), se la condizione è falsa viene eseguito il blocco "altrimenti" della condizione (blocco-2). Il blocco "altrimenti" è opzionale, cioè può essere omesso.

La selezione in Javascript/PHP si scrive usando il costrutto **if - else** (se-altrimenti):

```
if ( condizione ) {
    blocco-1
} else {
    blocco-2
}
```

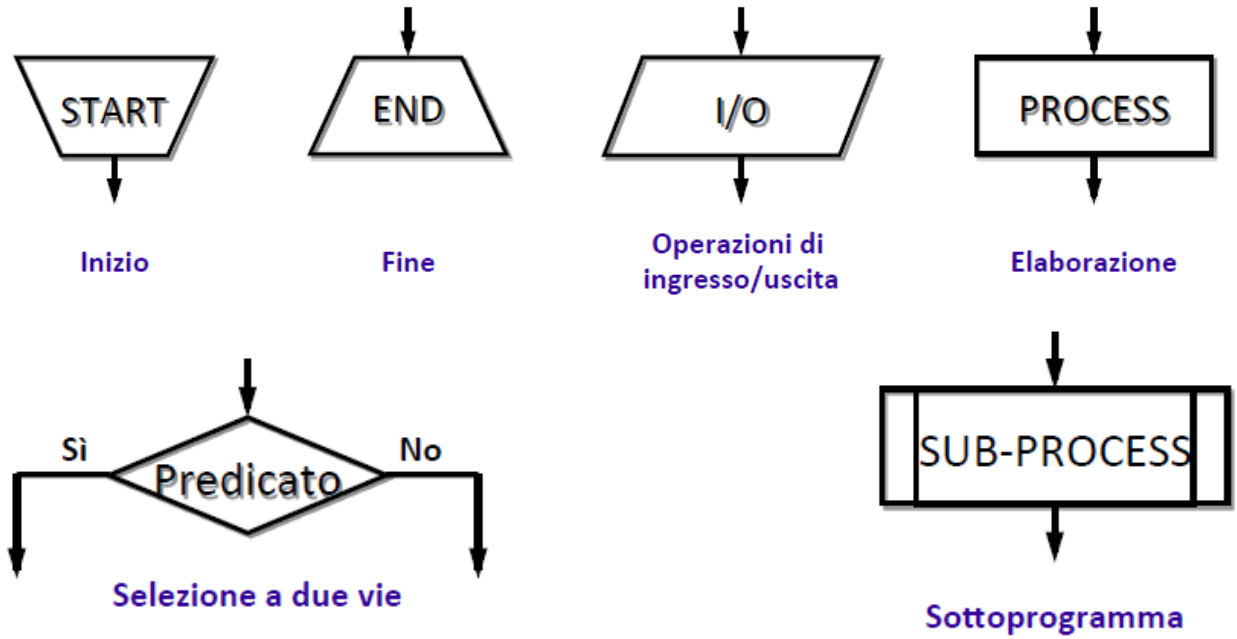
// punto di sincronizzazione: Qualunque sia il blocco eseguito (blocco-1 o blocco-2), al suo termine l'algoritmo prosegue dal medesimo punto di sincronizzazione

La parte "else + blocco-2" può anche non esserci: in quel caso, se la condizione è falsa, il programma non esegue nessuna istruzione e poi prosegue sempre a partire dal punto di sincronizzazione.

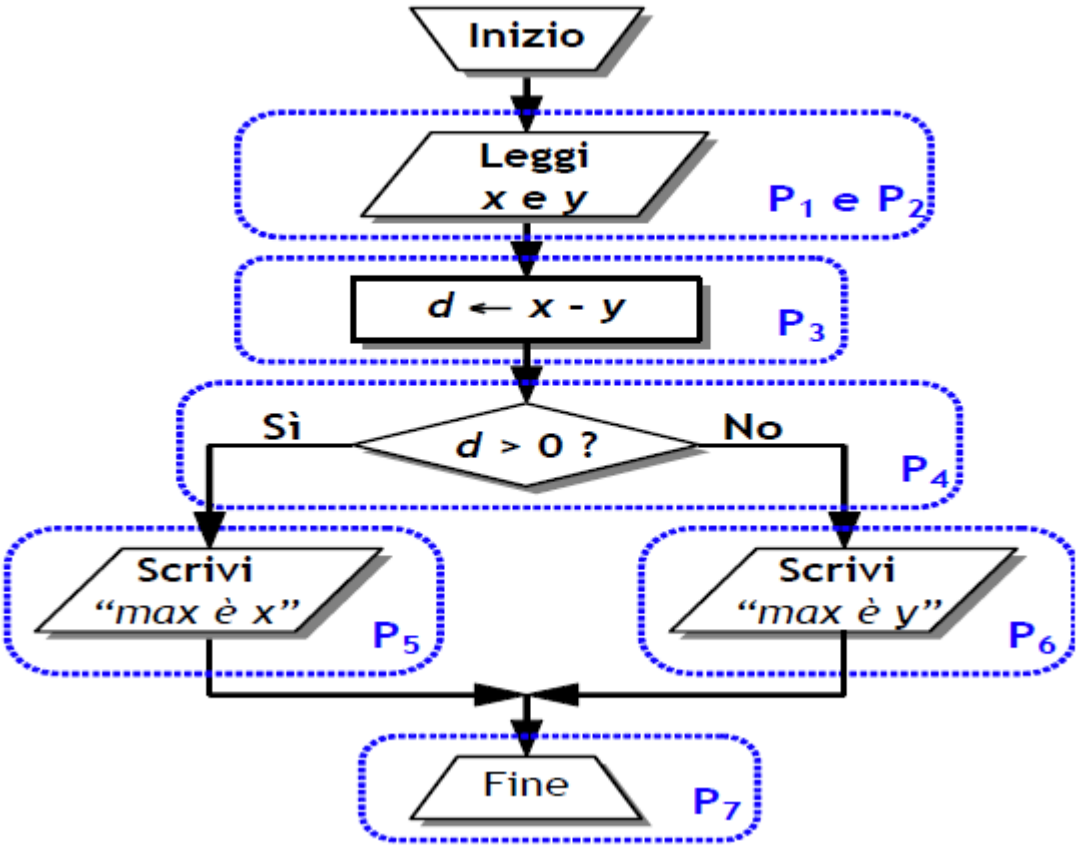
Esempio di programma in linguaggio naturale: Determinare il maggiore tra due numeri

P1 leggi un valore dall'esterno e assegnalo alla variabile x;
P2 leggi un secondo valore dall'esterno e assegnalo alla variabile y;
P3 calcola la differenza d fra x e y, cioè esegui d ← x – y;
P4 valuta se d è positivo: in caso *affermativo* prosegui con il passo P5, altrimenti (in caso *negativo*) salta al passo P7;
P5 scrivi "il numero maggiore è " seguito dal valore di x;
P6 salta al passo P11;
P7 valuta se d è nullo: in caso *affermativo* prosegui con il passo P8, altrimenti (in caso *negativo*) salta al passo P10;
P8 scrivi "i due numeri sono uguali";
P9 salta al passo P11;
P10 scrivi "il numero maggiore è " seguito dal valore di y;
P11 termina l'esecuzione.

Struttura dei diagrammi di flusso



Esempio di programma in diagramma di flusso: Ricerca massimo



Il costrutto ciclico: l'iterazione

Proviamo a pensare a un programma che deve stampare venti volte la stessa frase, oppure a un programma che deve leggere una serie di numeri e sommarli, fino a quando l'utente non scrive zero.

Con le istruzioni che abbiamo visto finora, nel primo caso sarebbe un programma molto lungo, nel secondo caso un programma impossibile.

I cicli o iterazione permettono invece di risolvere questo tipo di problemi – che richiedono di ripetere tante volte le stesse operazioni - in maniera molto semplice.

L'**iterazione** è l'atto di ripetere una serie di passi\operazioni con l'obiettivo di avvicinarsi a un risultato desiderato. Ogni ripetizione del processo di iterazione è essa stessa definita una iterazione e i risultati di una sono utilizzati come punto di partenza per quella successiva. Nell'ambito informatico, una iterazione o ciclo è la struttura di controllo che permette

all'algoritmo di ripetere un blocco di istruzioni, fino a quando una condizione risulta vera.

Per *blocco* e *condizione* valgono le definizioni date in precedenza.

Ad ogni passaggio, prima di eseguire il blocco, si controlla la condizione: *se è vera, si ripete il blocco; se è falsa, si prosegue con le istruzioni successive al ciclo.*

La forma algoritmica dell'iterazione (pseudo-codice):

finché (condizione) **ripeti** { blocco }

N.B. #1 La condizione di un ciclo deve poter diventare *falsa*, per cui le variabili all'interno del blocco del ciclo che modificano lo stato della condizione **devono essere modificate all'interno del blocco**.

N.B. #2 iterazione e ciclo sono sinonimi, in inglese viene chiamato loop.

Le iterazioni nei linguaggi di programmazione

L'iterazione in Javascript/PHP si scrive usando i costrutti **while** (finché), **do - while** (ripeti-finché), **for** (per tutti).

I cicli sopracitati sono composti da 3 elementi imprescindibili:

1. L'inizializzazione della variabile del ciclo
2. La condizione del ciclo basata sulla variabile inizializzata al punto 1
3. La modifica della variabile del punto 1 (nel for è l'incremento/decremento)

Esistono, in generale, due tipologie di ciclo: **scansione** e **ricerca**.

La *scansione* eseguirà il blocco del ciclo nella sua interezza: eseguirà **sempre** n iterazioni.

La *ricerca* eseguirà il blocco del ciclo fino a quando non sarà trovato l'elemento ricercato; eseguirà **al più** n iterazioni: nel caso migliore l'iterazione sarà una, nel caso peggiore saranno n.

Esempio in Pseudo-Codice: Prodotto di due numeri

```
Leggi alfa, beta;
prodotto = 0;
Finché alfa > 0 ripeti
    prodotto = prodotto + beta;
    alfa = alfa – 1;
stampa prodotto;
```

Il ciclo WHILE

Iniziamo a vedere la forma più semplice di iterazione presente nei linguaggi di programmazione, il *while*:

```
while ( condizione ) {
    blocco
}
```

Esempio di ciclo WHILE: Calcolare il fattoriale di 10

```
var numero = 10;
// la variabile numero contiene il valore da calcolare come fattoriale
var fatt = 1;
```

```
// La variabile fatt conterrà il risultato dell'operazione fattoriale, la inizializzo a 1 perché 1 è l'elemento neutro della moltiplicazione
while ( numero > 1 ) {
    fatt = fatt * numero;
    // Moltiplico il numero per il fattoriale
    numero  = numero - 1;  // Decremento il numero
}
// Quando la variabile numero conterrà 1, il ciclo while non verrà più eseguito perché la condizione al suo interno ( numero > 1 ) sarà falsa.
// All'interno della variabile fatt ci ritroveremo il fattoriale che stavamo cercando.
```

È come aver fatto:

fatt = (((((((((1 * 10) * 9) * 8) * 7) * 6) * 5) * 4) * 3) * 2)

Il Ciclo Do-While

Il do-while è un'alternativa al while che permette di effettuare il controllo della condizione alla fine del blocco anziché all'inizio. E' utile ad esempio quando si vuole controllare la correttezza dei dati inseriti.

```
do {
    blocco
} while ( condizione );
```

Il Ciclo FOR

Il *for* è un'alternativa al while che permette di inserire su un'unica riga **l'inizializzazione** del contatore, la **condizione** basata sul contatore e **l'incremento\decremento** del contatore. Si usa soprattutto quando il numero di ripetizioni da effettuare è predefinito.

for (*inizializzazione***;** *condizione***;** *incremento\decremento***) {**

    blocco

}

Esempio di ciclo FOR: Calcolare il fattoriale di 10

```
var numero = 10;
// la variabile numero contiene il valore da calcolare come fattoriale
var fatt = 1;
// La variabile fatt conterrà il risultato dell'operazione fattoriale, la inizializzo a 1 perché 1 è l'elemento neutro della moltiplicazione
for ( var i = 2; i <= numero; i = i + 1 ) {
    fatt = fatt * i;  // Moltiplico l'indice per il fattoriale
}
// Quando la variabile numero conterrà 11, il ciclo for non verrà più eseguito perché la condizione al suo interno ( i  <= numero ) sarà falsa.
// All'interno della variabile fatt ci ritroveremo il fattoriale che stavamo cercando.
```

È come aver fatto:

fatt = (((((((((1 * 2) * 3) * 4) * 5) * 6) * 7) * 8) * 9) * 10)

Strutture dati astratte: gli array/vettori

Un **vettore o array** è un insieme di variabili, tutte delle stesso tipo, che occupano uno spazio contiguo di memoria(RAM) e che hanno un nome composto da una prima parte comune e da una seconda parte (*indice*) specifica. Le variabili all'interno di un vettore vengono chiamate **elementi**.

Esempio: v[0] è il primo elemento(variabile) del vettore v e viene letto "v di 0".

L'**indice** di un vettore parte sempre da 0, se un vettore ha 10 elementi, l'ultimo elemento avrà indice 9.

I vettori offrono due grandi vantaggi:

1) Possibilità di creare tante variabili quante se ne vogliono con un'unica riga di istruzione

2) Possibilità di ripetere la stessa operazione su tutti gli elementi del vettore usando un ciclo for, ad esempio.

In Javascript la dichiarazione di un vettore è identica alla dichiarazione di qualsiasi altra variabile, l'inizializzazione viene effettuata attraverso l'uso delle parentesi quadre, inserendo al suo interno il numero di elementi che il vettore dovrà contenere:

<tipo> nome_vettore = [10];

V sarà un vettore di 10 elementi.

Javascript mette a disposizione una proprietà molto importante per il corretto utilizzo dei vettori, ovvero **length**. Per poter "accedere" a questa proprietà bisogna scrivere la seguente istruzione:

nome_vettore**.**length

Attraverso il simbolo **.** sarà possibile accedere a tutte le proprietà (e i metodi) del vettore.

Con **proprietà o attributi** s’intendono le caratteristiche di quella particolare struttura dati astratta.